Description

KEY-BASED ENCRYPTION

Technical field

[0001] The invention relates to encryption and more particularly to the renegotiation of keys used for encryption.

Background art

[0002] Individuals and businesses alike use computers to transmit and receive a variety of data. A reasonable proportion of such data is likely to be sensitive and thus ensuring data privacy is important.

[0003] One popular way of achieving data privacy is through the use of an encryption algorithm. Such algorithms are typically key-based and are either classed as symmetric or asymmetric.

[0004] Symmetric encryption algorithms make use of a secret key which is known only to the sender and the receiver of the data in question. The same secret key is used to encrypt the data at the sender as is used to decrypt the data when it is received by the receiver.

[0005] Asymmetric encryption algorithms on the other hand employ both a public key and secret key (secret key). The public key can be known to anybody whilst the secret key in known only to a limited number of entities. One key is used to encrypt the data, whilst the other key permits decryption of the data.

[0006] Secure Sockets Layer (SSL) is a protocol for achieving secure data transmission over the Internet. SSL employs both asymmetric and symmetric encryption techniques.

[0007] A pair of asymmetric keys is used in an initial authentication handshake between two parties (e.g. Alice and Bob). In the following example, Alice wishes to authenticate Bob (of course Bob may also want to authenticate Alice – this is advisable). Bob has a public-secret key pair. Bob's public key is disclosed to Alice. Alice transmits a message to Bob which Bob then encrypts with his secret key and returns to Alice. Alice decrypts the message from Bob using the public key that Bob disclosed to her earlier. If the decrypted message matches the message that Alice originally sent to Bob, then Alice can assume that Bob is who he says he is. SSL however also uses digital signatures to prevent a third party from obtaining Alice's original message and impersonating Alice. SSL also

makes use of certificates. A certificate is used to prove that a public key really does come from, for example, Bob.

[0008] Having authenticated Bob, Alice is now ready to exchange data with Bob. Before data exchange can however take place, Alice and Bob must agree upon a symmetric (secret) key. The data to be exchanged is first encrypted with this secret key. Since both parties have agreed upon the secret key, Alice can encrypt her data using the key whilst Bob can decrypt data received from Alice.

[0009] It will be appreciated that if the secret key is discovered by an unauthorised third-party, it can be used to decrypt data and to encrypt spoof messages/alter data.

[0010] It is for this reason that it is preferable to periodically renegotiate the SSL secret key used by Alice (client) and Bob (server) to exchange data. The renegotiation of the secret key involves performing a CPU intensive handshake on both the client and server. This is especially processor intensive when each renegotiation involves a full asymmetric authentication followed by the negotiation of a symmetric secret key.

[0011] Note, a detailed overview of SSL can be found at http://developer.netscape.com/tech/security/ssl/howitworks.html.

[0012] Current Solutions

[0013] Current secret key renegotiation implementations generally use one of two methods:

[0014] (i) timed reset in which a renegotiation is initiated by the SSL client every x minutes (e.g. a web browser may initiate a key renegotiation every 2 minutes); or

[0015] (ii) initiation after a certain threshold of bytes has flowed.

[0016] These solutions do not however work efficiently in a messaging environment since in such an environment a communications link typically fluctuates between idle and busy. The solutions mentioned above are especially inefficient where a communications link is particularly idle or busy at varying times of day (as is likely in a messaging environment).

[0017] Problems with Current Solutions

[0018] (i) timed renegotiation – idle communications link

[0019] There could be an unnecessary number of full authentications and renegotiations when no data (messages) has been sent over the communications link for a prolonged period of time. In other words, a client may needlessly often authenticate a server with which it wants to communicate and then renegotiate a

secret key with that server. Thus performance is degraded unnecessarily.

[0020] (ii) byte threshold implementation – idle communications link

[0021] This solution increases the period of time in which a secret key is valid on an idle link and gives a hacker more time to attack the secret key and send 'spoof' messages without detection.

[0022] (iii) timed negotiation – busy communications link

[0023] A busy communications link will flow a large amount of data encrypted with the same secret key. Whilst the hacker is unlikely to be able to break the key in the sort of times generally used for timed renegotiation, the problem however is that he will be able to record the encrypted data and then analyse it at his leisure. This is easier for him if large amounts of the data are encrypted with the same secret key and thus the security of a large amount of data may be compromised using this solution when the communications link is busy.

[0024] (iv) byte threshold implementation – busy communications link

[0025] The amount of data encrypted with the same secret key on a busy communications link will be minimised. Thus this solution minimises the amount of data encrypted with a single secret key. However this solution is not good, when the link is predominantly idle (see above).

[0026] Thus encryption in an environment that fluctuates between being busy and being idle has thus far been problematic.

Disclosure of the invention

[0027] According to one aspect there is provided a method for facilitating secure data communications using a secret key for encrypting data flowing between first and second entities over a communications link, the method comprising: determining that the communications link has been idle; determining that there is now data to flow over the previously idle communications link; and responsive to determining that there is now data to flow over the previously idle communications link, initiating generation of a new secret key, the new secret key for encrypting data sent between the first and second entities over the communications link.

[0028] In this way key generation only occurs when transmission over an idle communications link is about to recommence. This is contrary to the prior art in which key generation may occur on a timed basis.

[0029] Preferably it is also possible to determine when a preconfigured amount of data has been sent over the communications link. Once the preconfigured amount

has been sent over the link, then generation of a new secret key is preferably initiated.

[0030] This caters for the situation in which the communications link is not predominantly idle. Thus even on a busy link, key generation occurs at frequent enough intervals.

[0031] In one embodiment the communications link must have been idle for at least a predetermined amount of time before the generation of the new secret key is initiated as a result of a determination that there is now data to flow over the link.

[0032] In this way, short periods of idleness do not immediately cause the initiation of the process for generating a new secret key.

[0033] Note, a simple timeout system could be used which says that if the link has been idle for at least x seconds and there is now data to flow, then a new secret key should be generated.

[0034] In a preferred embodiment, once it is determined that the communications link has been idle for a predetermined period of time, the second entity is informed that the first entity is still present via a heartbeat.

[0035] In this way, the second entity is aware that the first entity is still alive even though the second entity currently has no data to flow over the communications link. Note, the first entity may send more than one heartbeat to the second entity (i.e. if the link is idle for long enough).

[0036] The second entity preferably confirms receipt of a heartbeat from the first entity.

[0037] In one embodiment, if no confirmation of receipt of a heartbeat is received by the first entity within a predetermined amount of time, then communication by the first entity with the second entity is terminated. This is because either the second entity has failed, or a third party is consuming the heartbeats / responses to the heartbeats.

[0038] In another embodiment, if no confirmation of receipt of a heartbeat is received by the first entity within a predetermined amount of time, then the generation of a new secret key is initiated before it is permissible for data to be transmitted again by the first entity to the second entity. Of course, unless the process also includes authentication (see later), a third party could pretend to be the second entity and thus be involved in the key generation process.

[0039] In a preferred embodiment, it is possible to determine that the communications link has been idle enough to cause the first entity to send a heartbeat to the

second entity. Preferably responsive to determining that the link has been idle enough to cause the first entity to send a heartbeat to the second entity, generation of a new secret key is initiated.

[0040] In a preferred embodiment, authentication of at least the second entity is initiated prior to initiation of generation of a new secret key.

[0041] Generation of a new secret key is preferably as a result of a negotiation process carried out between the first and the second entity.

[0042] According to another aspect, there is provided a method for facilitating secure data communications using a secret key for encrypting data flowing between first and second entities over a communications link, the method comprising: determining that the communications link has been idle; and responsive to determining that the communications link has been idle, ignoring data encrypted with the secret key.

[0043] Preferably only subsequent data encrypted with a newly generated secret key is accepted.

[0044] Preferably the communications link must be idle for at least a predetermined amount of time. Preferably this is indicated via the receipt of a heartbeat from the first entity.

[0045] In accordance with one embodiment, when it is determined that the communications link has been idle for at least a predetermined amount of time and that no heartbeat has been received from the first entity, communication with the first entity is terminated.

[0046] This is because it is assumed that either the first entity has failed or that a third-party is consuming the heartbeats.

[0047] In another embodiment, responsive to determining that the communications link has been idle for at least a predetermined amount of time and that no heartbeat has been received from the first entity, only subsequent data encrypted with a newly generated secret key is accepted.

[0048] According to another aspect, there is provided an apparatus for facilitating secure data communications using a secret key for encrypting data flowing between first and second entities over a communications link, the apparatus comprising: means for determining that the communications link has been idle; means for determining that there is data to flow over the previously idle communications link; and means, responsive to determining that there is data to

flow over the previously idle communications link, for initiating generation of a new secret key, the new secret key for encrypting data sent between the first and the second entities over the communications link.

[0049] According to another aspect, there is provided an apparatus for facilitating secure data communications using a secret key for encrypting data flowing between a first and a second entity over a communications link, the apparatus comprising: means for determining that the communications link has been idle; and means, responsive to determining that the communications link has been idle, for ignoring data encrypted with the secret key.

[0050] Preferably data encrypted with the secret key is ignored in the sense that it is not considered safe to trust the integrity of the data. Thus preferably only data encrypted with a newly generated secret key is accepted as trustworthy.

[0051] It will be appreciated that the invention may be implemented in computer software.

Brief description of the drawings

[0052] A preferred embodiment of the present invention will now be described, by way of example only, and with reference to the following drawings:

[0053] Figure 1a illustrates a client-server component diagram in accordance with a preferred embodiment of the present invention; and

[0054] Figure 1b is a flow chart of the processing performed by the client in accordance with a preferred embodiment of the present invention.

Mode(s) for carrying out the invention

[0055] A preferred embodiment of the present invention will now be described with reference to figures 1a and 1b. The two figures should be read in conjunction with one another.

[0056] An SSL client 5 desires to transmit data to an SSL server 6. First the SSL client initiates a connection with the server via communications link 90 using connection initiator 55. Client 5 then authenticates server 6 using authenticator 10 which communicates with an equivalent component 10' on the server (step 100).

[0057] Having authenticated the server 6, the client and server negotiate a symmetric secret key via key negotiator components 20, 20' (step 110). This secret key is subsequently used to encrypt and decrypt messages that the client flows across the communications link 90.

[0058] A data detector 70 on the client works to detect whether the client 5 has any data to flow over communications link 90 (step 120). If there is data to flow over the link, then it is detected at step 130 whether the link was previously idle enough to result in the client sending a heartbeat to the server (see later).

[0059] Assuming that this is not the case then this data is encrypted with the current secret key and is sent (not shown). It is determined via byte measurer 50 (step 150) whether a pre-configured number of bytes have been sent. If the answer is no, then the process loop round to step 120 to see whether there is more data to be flowed.

[0060] If a pre-configured number of bytes has been sent (as detected by byte measurer 40), then it is time to re-authenticate and re-negotiate the key (steps 100, 110) using components 10, 10', 20, 20'. At this point a number of bytes sent value held by byte measurer 40 is reset to zero.

[0061] The configurable byte threshold ensures that the amount of data sent on a busy communications link with the same secret key is limited since the secret key will be renegotiated regularly as a result of the byte threshold being met. Thus the amount of data encrypted with the same secret key is minimised.

[0062] Note the setting of an appropriate byte threshold is a trade-off:

[0063] The lower the threshold, the more often re-authentication is performed and the secret key is changed - thus the more processing power involved. However the more often re-authentication is performed and the secret key is changed, the more secure the data flowing over the communications link is; and

[0064] The higher the threshold is, the better the performance (due to fewer re-authentications and secret key renegotiations). Of course the data flowing over the communications link is less secure than in an environment in which the threshold is lower.

[0065] A timer 30 is used by the data detector component 70 to determine when the communications link 90 has been idle for a configurable period of time. If this is the case, it issues (via the heartbeat issuer 50) a special 'heartbeat' message to confirm to the SSL server 6 that it is still present (step 160). (The timer is then reset to zero – note the timer is also preferably set to zero when re-authentication is initiated). The client waits for a reply to the heartbeat (step 170) from the server (heartbeat receiver 75, heartbeat reply generator 80) – see later.

[0066] Note, the configurable period of time is preferably not too short (e.g. 10 seconds)

since this could result in numerous heartbeats (i.e. too much unnecessary traffic). The time chosen depends on the environment – a period of 5 minutes might for example be appropriate.

[0067] After the SSL server has received one or more 'heartbeat' messages it will reject as ‘spoofs’ any other messages containing application data encrypted with the same secret key (data rejecter component 95). Upon detection of spoof data an appropriate action(s) should then be taken such as logging this with an administrator and closing the connection with the client.

[0068] For the SSL client to send a new message to the SSL server (which has received a heartbeat indicating that the communications link was previously idle) it must first renegotiate a new secret key before sending the message to avoid the SSL server rejecting it as a 'spoof' (steps 120, 130, 100 and 110). Thus there should be no security exposure after a period of idleness.

[0069] As discussed above, spoof data sent after a period of idleness preferably causes the server to terminate the connection with the client. The client can then choose to resume its connection with the server and has to re-authenticate and re-negotiate prior to sending any more data to the server.

[0070] Note, since heartbeats do not contain any useful data, they do not have to be encrypted.

[0071] If the SSL server does not receive a 'heartbeat' when it detects (via data detector component 70’ and timer component 30’) that the link has been idle for greater than the configurable period of time (i.e. the same period of time as used by client 5), then the SSL server closes its connection via connection terminator 85. This prevents a hacker from consuming 'heartbeat' messages to prevent a secret key renegotiation to extend the lifetime of the secret key.

[0072] Note, there is no need to detect spoof heartbeats since they do not compromise any application data.

[0073] If the SSL server 6 is present and has received a heartbeat from the client then it replies to the special 'heartbeat' message and remembers that the connection has been idle for long enough for a heartbeat to flow across the communications link. The SSL client can send any number of 'heartbeat' messages to confirm that it is still present.

[0074] Note the 'heartbeat' message does not contribute to the byte total used in calculating when a secret key should be triggered by the byte threshold.

[0075] Once the client has received a reply from the server, the process of figure 1b loops round to step 120. If it is determined that there is now data to flow over the link, then it is again tested at step 130 whether the link was previously idle enough to cause the generation of a heartbeat. If the answer is yes, then the secret key should be renegotiated. The client will not therefore send any more data to the server until re-authentication and key negotiation has been effected.

[0076] This means that if a hacker has managed to break the secret key due to the prolonged period during which the communications link was idle, that key is now no use to them.

[0077] If no reply is received then the client closes its connection using connection terminator 55 (step 180). This is because a failure to reply indicates that either the server is no longer present or someone else is consuming the server's replies.

[0078] Note, in an alternative embodiment, the client may attempt to contact the server an additional number of times before terminating the connection. This is because the lack of response from the server could just be a temporary problem. To be on the safe side, re-authentication/key negotiation could be initiated.

[0079] Note, the timing between heartbeats (when more than one is sent on an idle link) is preferably constant. If random timing is used between each heartbeat message, then it would not be possible to predict when a hearbeat was overdue (quite possibly consumed by a hacker).

[0080] It will of course be appreciated that the time before a heartbeat is first sent (and intervals between heartbeats) and byteflow counts are preferably chosen so that the same secret key does not remain in use for a prolonged period. Note, a 'spoof' message could still be achievable if the chosen values are high enough to provide a hacker with the time to capture and discover the secret key. However once a heartbeat triggered secret key renegotiation occurs, the hacker will be unable fool the server any longer. For this reason, it is preferably for the data sending side to initiate the negotiation of a new key. Otherwise, if the server is receiving spoof messages correctly encrypted, from the server's point of view re-negotiation is not required.

[0081] There are four key advantages of using the solution thus far described;

[0082] (i) The proposal ensures that re-authentication and renegotiation of the secret key is only issued when absolutely necessary on idle communications

links to achieve optimum performance whilst remaining secure.

[0083] (ii) The ability to detect a 'spoof' message even though it has been encrypted with the correct secret key is provided through the use of ' heartbeat' messages – since the secret key is renegotiated when data communication is resumed;

[0084] (iii) The proposal ensures the secret key is changed regularly on a busy communications link to limit the amount of application data that could be read by a hacker with a compromised secret key; and

[0085] (iv) The special 'heartbeat' messages do not contain application data and hence are useless to a hacker even though the secret key used to encrypt & decrypt the data could be discovered through brute force

[0086] This protocol preferably ensures that authentication and key negotiation is always carried out after the communications link has been idle for a period long enough such that one or more heartbeats have flowed across the link and when a certain number of bytes have flowed across the communications link.

[0087] A hacker sending a 'spoof' message, even though he may have discovered the current agreed secret key, cannot follow the agreed protocol as he does not possess the asymmetric secret key for initiating a re-authentication (of the client) and key negotiation (neither does he have the certificate for re-authentication). Further the old symmetric secret key is invalid from the moment the server sees a heartbeat from the client.

[0088] This solution effectively prevents a hacker from sending 'spoof' messages on an idle communications link without the need to unnecessarily perform re-authentication, key negotiation. This solution also copes with a busy communications link.

[0089] Note it is possible to configure SSL such that the SSL client doesn't need to present authentication information to the SSL server in order for a successful re-authentication to take place – in this case only the server is authenticated to the client, not the other way round. However this is not advisable in a secure peer-to-peer environment since it would enable a third-party to pretend to be the client and to communicate with the server as such.

[0090] Note, whilst the present invention has been described as particularly applicable to the messaging environment, no limitation to such is intended. The invention is applicable to any environment which fluctuates between idle and busy periods.

[0091] Further, whilst the invention has been described in terms of the SSL encryption

protocol, once again no such limitation is intended. The invention is however particularly applicable to any environment where authentication and key negotiation is processor intensive. Another example is TLS.

[0092] Note, in the exemplary embodiment, data is flowing from the client to the server. This does not have to be the case – data may flow in the opposite direction. Preferably whoever is sending data initiates authentication and key negotiation and also sends heartbeats.

[0093] In an alternative embodiment, authentication and key renegotiation is always initiated by the SSL client. Thus if the SSL server has data to send, the server asks the SSL client to authenticate and renegotiate first. The opposite could also be true.

[0094] Note, whilst the preferred embodiment has been described in terms of performing on each occasion an initial full handshake (asymmetric authentication) and then negotiation of the secret key, this does not have to be the case. The invention is especially applicable in this situation since authentication followed by key negotiation is particularly processor intensive. However the invention is also preferably applicable in an environment which uses session caching (less processor intensive). This is a feature that is available in, for example, SSL v3.0 and TLS.

[0095] Session caching can be carried out during the initial handshake. The client and the server save a common session ID, the master secret key and some certificate chains. This information is held in a cache usually for a configurable period of time.

[0096] If a subsequent handshake is requested (i.e. when the client requests a new secret key) and this information has not expired from the cache, both sides present each other with their session IDs. If the session IDs match then the cached information will be used to reduce the processing carried out during the handshake - this is commonly referred to as an abbreviated handshake as opposed to a full handshake.

[0097] Note, the weakness with using session caching is that a hacker need only present the original session ID when responding to a handshake (no certificates are swapped and no public key operations take place). The session ID is included in a client “hello” flow so could be snooped off the wire.

[0098] Note, data does not have to flow in one direction only – data may flow in both

directions. In this scenario, whoever has data to send when secret key renegotiation becomes necessary preferably initiates the secret key renegotiation. One of the two ends is preferably specified as being responsible for sending heartbeats (i.e. after no data has flowed in either direction for at least a predetermined amount of time). Heartbeats and responses thereto are thus used to determine the presence of both ends. The byte count used is preferably the sum total of all data sent over the communications link during a particular time period – i.e. includes data sent by both ends. In one embodiment, one end keeps track of the byte count and the idleness of the link and informs the other end when either of the two thresholds are met.